

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning documents *will not* correct images,  
please do not report the images to the  
Image Problem Mailbox.**

EL 96141475E

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

Destination Application Program Interfaces

Inventor(s):

Rebecca C. Weiss

Dale A. Sather

Mei L. Wilson

Kirt A. Debique

Alexandre V. Grigorovitch

Robin C.B. Speed

Adil Sherwani

ATTORNEY'S DOCKET NO. MS1-1722US

## **RELATED APPLICATIONS**

This application is related to the following application, the disclosure of which are incorporated by reference:

- U.S. Patent Application No. [Attorney Docket No. MS1-1724US] entitled "Media Processing Methods, Systems and Application Program Interfaces", filed December 8, 2003;

## **TECHNICAL FIELD**

This invention relates to media processing methods, systems and application program interfaces.

## **BACKGROUND**

As multimedia systems and architectures evolve, there is a continuing need for systems and architectures that are flexible in terms of implementation and the various environments in which such systems and architectures can be employed. As an example, consider the following as flexibility is viewed from the vantage point of software applications that execute in conjunction with such multimedia systems.

When it comes to rendering multimedia presentations, some software applications are very basic in terms of their functionality. That is, these basic types of applications might simply wish to provide a multimedia system with only a small amount of data that pertains to the presentation and have the multimedia system itself do the remainder of the work to render the presentation. Yet other more complex types of application may wish to be more intimately involved with the detailed processing that takes place within the multimedia system.

1       Against this backdrop, there is a continuing need to provide multimedia  
2 systems and architectures that meet the needs of applications that are distributed  
3 along a spectrum of simple applications to complex applications.

## 4 5 **SUMMARY**

6       Media processing methods, systems and application program interfaces  
7 (APIs) are described. In but one embodiment, a destination component, also  
8 referred to as a destination, provides an application a simple and unified way of  
9 describing where and how to do rendering, archiving, broadcasting (or other types  
10 of media output processing) of media from an origin to an output target of choice  
11 (also referred to herein as a media sink), without requiring the application to have  
12 intimate knowledge about underlying components, their connectivity and  
13 management. For example, applications can use a destination to help manage the  
14 rendering or archiving (or other processing) of the particular media, which  
15 advantageously allows the application to simply provide the media (or its location)  
16 and identify where the media is rendered, archived, multicasted, etc.

## 17 18 **BRIEF DESCRIPTION OF THE DRAWINGS**

19       Fig. 1 is a block diagram of a system in accordance with one embodiment.

20       Fig. 2 is a flow diagram that describes steps in a method in accordance with  
21 one embodiment.

22       Fig. 3 is a block diagram of a system in accordance with another  
23 embodiment.

24       Fig. 4 is a flow diagram that describes steps in a method in accordance with  
25 another embodiment.

1 Fig. 5 is a flow diagram that describes steps in a method of changing a  
2 destination, in accordance with another embodiment.

3 Fig. 6 is a diagram that illustrates interfaces of a destination in accordance  
4 with one embodiment.

5 Fig. 7 is a diagram that illustrates interfaces of a destination collection in  
6 accordance with one embodiment.

## 7 8 DETAILED DESCRIPTION

### 9 Overview

10 Fig. 1 illustrates a high level block diagram of an exemplary system in  
11 accordance with one embodiment, generally at 100. In system 100, an application  
12 102 interacts with a media engine component (or more simply a media engine  
13 104), a destination 116, content 118 (typically any suitable type of multimedia  
14 content, including for example stored or “live” video and/or audio streams) and  
15 one or more output targets 122 (e.g., a window, a file, a broadcast publishing point  
16 and the like).

17 In at least some embodiments, application 102 can use system 100 to  
18 participate in a *presentation*. A presentation, as used in this document, refers to or  
19 describes the handling of media content. Thus, a presentation can include visually  
20 and/or audibly presenting media content, such as a multimedia presentation in  
21 which both audio and accompanying video is presented to user via a window  
22 executing on a display device such as a display associated with a desk top device.  
23 A presentation can also refer to writing media content to a computer-readable  
24 medium such as a disk file. Further, a presentation can refer to broadcasting or  
25

1 multicasting media content over a network. Thus, a presentation is not simply  
2 limited to rendering multimedia content on a computing device.

3 In at least one embodiment, a destination is an object that defines output  
4 target(s) 122 where a presentation is to be presented (e.g. a window, disk file,  
5 URL and the like) and what the presentation does (e.g., render media, archive a  
6 media file, broadcast or multicast media over a network, and the like).

7 In at least one embodiment of system 100, application 102 creates  
8 destination 116 and media engine 104. After the application provides media  
9 engine 104 with destination 116 and some information describing from where the  
10 multimedia data for content 118 should be sourced, media engine 104 can include  
11 a media source 108, transforms 110 and one or more media sinks (e.g., media  
12 sinks 112 and 114 in this example), as described in the aforementioned co-pending  
13 U.S. Patent Application No. [Attorney docket No. MS1-1724US] entitled "Media  
14 Processing Methods, Systems and Application Program Interfaces".  
15 Destination 116 is responsible for providing media sinks 112 and 114 to the media  
16 engine.

17 In one embodiment, media source 108 comprises a component that can be  
18 used to read a particular type of media content from a particular source. For  
19 example, one type of media source might read compressed video data and another  
20 media source might read compressed audio data. The data can be received from a  
21 data store or from a device capturing "live" multimedia data (e.g., a camcorder).  
22 Alternately or additionally, a media source might be used to read a compressed  
23 data stream and separate the data stream into its compressed video and compressed  
24 audio component. Alternatively or additionally, a media source might be used to  
25 receive compressed data from over the network.

1 In this embodiment, transforms 110 comprise any suitable data handling  
2 components that are typically used in presentations. Such components can include  
3 those that uncompress compressed data, compress uncompressed data, and/or  
4 operate on data in some way, such as by imparting an effect to the data, as will be  
5 appreciated by the skilled artisan.

6 Media sinks 112 and 114 are typically associated with a particular type of  
7 media content and presentation. Thus, audio content might have an associated  
8 audio sink for playback such as an audio renderer. Likewise, video content might  
9 have an associated video sink for playback such as a video renderer. Additional  
10 media sinks can archive multimedia data to computer-readable media, e.g. a disk  
11 file, a CD, or the like. Additional media sinks can send data over the network.

12 In simple scenarios, a presentation may include only a single media sink  
13 (e.g., for a simple audio-only playback presentation the media sink may be an  
14 audio renderer). In the exemplary system of Fig. 1, media sinks 112 and 114 may  
15 be, for example, audio and video renderers, respectively, to play audio/video  
16 streams received from a camcorder device.

17 One reason for providing destination 116 is to abstract away the specific  
18 details of media sinks 112 and 114 from application 102. That is, in one  
19 embodiment, application 102 need only to provide information about multimedia  
20 content (in one embodiment, an URL indicating the location of the content) and  
21 information about destination 116 to media engine 104 (e.g., a pointer to the  
22 location where destination 116 can be accessed) to create an appropriate  
23 presentation at output target 122. After providing the content location information  
24 and the destination location information, destination 116 manages the operations  
25

1 needed for setting up the outputs of the presentation so that the application no  
2 longer needs to participate in the presentation.

3 In another aspect, implementations of destination 116 are based on an open  
4 and extensible architecture so that users can specify new destinations as existing  
5 applications evolve and/or new applications are written. Fig. 6 illustrates some of  
6 the APIs that can be implemented by destination 116. For example, the interface  
7 IMFDestination shown in Fig. 6 can include the following non-exclusive methods.

### 8 GetOutputInfo

9 The GetOutputInfo method is called by media engine 104 once for each  
10 input stream from the media source in the presentation. The input arguments  
11 include a pointer to a presentation description of the input stream and a pointer to  
12 a media type that will be used for the input stream. The output argument can be  
13 either an output info object (implementing the IMFOutputInfo interface) or a  
14 collection of output info objects. From an output info, the following objects can  
15 be obtained: an output, a property store, and a presentation clock. The “output” is  
16 either a media sink or it is an object implementing the IActivate interface, which is  
17 an object that is used by the media engine to create a media sink. A property store  
18 contains properties that may be helpful to the media engine in figuring out what to  
19 do with this output; it can include but is not limited to information such as: a  
20 stream sink ID, which is an identifier for the stream; an output ID, which is used to  
21 identify the output; and a directive, which indicates whether the media sink should  
22 be shut down when the media engine is finished using the media sink. A  
23 presentation clock if used is provided for each stream of the presentation. A user  
24 of application 102 can use a presentation clock to control a media sink  
25

1 independently of the other media sinks of the presentation. This method is  
2 described further below.

### 3 Lock/UnlockDestination

4 The Lock/UnlockDestination methods are used by the media engine to  
5 synchronize access to the destination while starting a new presentation. For  
6 example, a presentation may include multiple streams requiring multiple  
7 GetOutputInfo calls. The media engine can call the LockDestination method  
8 before starting this set of GetOutputInfo calls and call the UnlockDestination  
9 method after all GetOutputInfo calls for a presentation are complete. This method  
10 is described further below.

### 11 Example of Opening a Presentation using a Destination

12 Fig. 2 illustrates an example operational flow of system 100 (Fig. 1)  
13 according to one embodiment. Referring to Figs. 1 and 2, application 102 can  
14 open a presentation as follows. In a block 202, application creates media  
15 engine 104. In one embodiment, application 102 creates media engine 104  
16 (described in the aforementioned co-pending U.S. Patent Application No.  
17 [Attorney docket No. MS1-1724US] entitled "Media Processing Methods,  
18 Systems and Application Program Interfaces"). In one embodiment, the operating  
19 system includes a standard media engine that can be used to implement media  
20 engine 104.

21 In a block 204, application 102 creates destination 116. In one  
22 embodiment, the operating system includes standard destination objects for  
23 common scenarios (e.g., playing back audio and/or video content in a window or  
24 archiving media in a file to a certain multimedia format, etc.). In other  
25

1   embodiments, application 102 includes “custom” destination objects that conform  
2   to a publicly available destination specification.

3         In a block 206, application 102 causes media engine 104 to open a  
4   presentation. In one embodiment, application 102 makes an “Open” call  
5   (examples of which are described below) to an API of media engine 104,  
6   providing location information for the media content and the destination created in  
7   block 204. For example, the “Open” call can be to an “OpenURL” method to  
8   open a URL for playback scenarios, or an “OpenSource” method to open an  
9   already-existing media source. In some embodiments, such “Open” methods have  
10   at least two arguments, namely, pointers to the content and the destination. Other  
11   “Open” methods can also include a pointer to descriptor information (e.g., a  
12   topology indicating compression and bit-rate information for encoding scenarios)  
13   as an argument. Embodiments of various “Open” methods are described below in  
14   more detail. In performing the “Open” method, media engine 104 makes a  
15   GetOutputInfo call to an API of destination 116 for each stream of the  
16   presentation.

17         In block 208, in one embodiment, media source 108 provides to media  
18   engine 104 description information regarding each media stream of the  
19   presentation. In block 209, in one embodiment, the media engine passes  
20   descriptor and media type information of each stream generated in block 208 to  
21   destination 116. In block 210, in one embodiment, destination 116 provides an  
22   output that will result in the stream being presented in output target 122 for each  
23   stream. This information allows media engine 104 to set up transforms 110 and  
24   obtain media sinks (e.g., media sinks 112 and 114) used in performing the  
25   presentation. In one embodiment, destination 116 performs the aforementioned

1   GetOutputInfo method to provide the information in block 210. In this  
2   embodiment, in performing the GetOutputInfo method, destination 116 outputs:  
3   (1) a property store; and (2) an output info object (also referred to herein simply as  
4   an output info) or a collection of output infos.

#### 5       Output info

6       Because the GetOutputInfo method is called for each stream in the  
7   presentation, each stream has one or more associated output infos unless the  
8   destination does not have an output for the stream. In one embodiment, the output  
9   info object encapsulates three pieces of data: an “output” object, a property store,  
10   and a presentation clock. In this embodiment, only the output info object (or more  
11   simply “output info”) is mandatory. These data from the output info are described  
12   below.

#### 13       Output

14       In one embodiment, media engine 104 uses each output info to obtain an  
15   output object. What this object is can be determined at run-time; it may itself be a  
16   media sink, or it may be an IActivate from which a media sink can be obtained.  
17   The media engine obtains and configures the corresponding media sink (e.g.,  
18   audio and/or video renderers) for the associated stream as described in the  
19   aforementioned co-pending U.S. Patent Application No. [Attorney Docket No.  
20   MS1-1724US] entitled “Media Processing Methods, Systems and Application  
21   Program Interfaces”).

#### 22       Property Store

23       In some embodiments, the output info object provided by the  
24   destination 116 for a stream may include a property store. Properties in that  
25   property store may include but are not limited to: identifier information indicating

1 which stream sink to use from the media sink, and an identifier for the output info  
2 that will assist the media engine in handling destination changes, and/or a directive  
3 indicating whether to shut down the media sink associated with the output info  
4 (i.e., media engine 104 obtains media sinks from output infos, as described in the  
5 aforementioned co-pending U.S. Patent Application No. [Attorney docket No.  
6 MS1-1724US] entitled “Media Processing Methods, Systems and Application  
7 Program Interfaces”). In some scenarios, the property store can contain nothing.

#### 8 Presentation Clock

9 In some embodiments, the output info object provided by the destination  
10 116 for a stream may include a presentation clock. A presentation clock allows  
11 application 102 to independently control (e.g., start, stop, pause, etc.) the media  
12 sink obtained from the associated output. For example, in an encoding scenario,  
13 the application may be previewing a video stream while the stream is being  
14 archived. The application’s user may then be able to stop archiving the stream  
15 (via the presentation clock) when the user detects an unwanted portion of the  
16 stream (e.g., commercials). The application’s user can then restart archiving of the  
17 stream when the unwanted portion ends.

#### 18 Custom and Standard Destinations

19 Fig. 3 illustrates an example system 300 that uses destinations, according to  
20 another embodiment. System 300 is substantially similar to system 100 (Fig. 1)  
21 except that system 300 includes one of either custom destination 316 or standard  
22 destination(s) 316A instead of destination 116 (Fig. 1) built for a software  
23 environment 302 referred to herein as Media Foundation. In this embodiment,  
24 application 102 is built on top of Media Foundation and is able to create:  
25 (1) media engine 104 (e.g., a standard media engine that is part of Media

Foundation); and (2) a destination (e.g., a custom destination 316 written for application 102 conforming to a media engine specification or a standard destination 316A that is part of Media Foundation). If the application is using one of the Destinations provided by Media Foundation, a destination as can be created by using the following non-exclusive Media Foundation methods:

#### MFCreatPlaybackDestination

The method MFCreatPlaybackDestination is called by an application to create a standard playback destination. In one embodiment, this method has, as one input argument, a handle to a window in which playback of video content is to take place. Optionally, the method can include a pointer to a presentation clock (described above). In one embodiment, for audio-only destinations, the handle can be omitted (i.e., set to zero or null). The destination will then provide the media engine with output infos from which the appropriate media sinks can be created; in this case, these are a video renderer to render into the specified window, and an audio renderer that will render audio into the system's default speakers. This method is described further in an API section below.

#### MFCreatXXXArchiveDestination

The MFCreatXXXArchiveDestination methods (where XXX represents a code for one of the various types of media formats (e.g., ASF, WAV and others) supported by Media Foundation. For example, MFCreatASFArchiveDestination can be called by an application to create a standard ASF encoding destination. In one embodiment, this method has, as one input argument, a pointer to a file in which the media stream is to be archived. This method has a presentation descriptor for another input argument, which includes information such as the compression, bit-rate, etc. Optionally, the method can include a pointer to a

1 presentation clock (described above). This method is described further in an API  
2 section below.

### 3 MFCCreateMediaSinkDestination

4 The method MFCCreatePlaybackDestination is called by an application that  
5 has created its own media sink.

### 6 Destination Collections

7 In some scenarios, the application may wish to use a combination of  
8 destinations. For example, in a typical encoding scenario, the application may  
9 wish to display the content while it is being encoded. Thus, two destinations are  
10 needed. In one embodiment, the application can create a destination collection  
11 object that includes two “sub-destinations”- one for the encoding destination and  
12 another for the preview destination. In one embodiment, the destination collection  
13 object (or more simply a destination collection) also is able to send a  
14 MEDestinationChanged event, which can be sent to the media engine to indicate if  
15 the set of outputs for the destination changes. Note, the MEDestinationChanged  
16 event is not limited to destination collections. In one embodiment based on Media  
17 Foundation, a destination collection object as shown in Fig. 7 can be created by  
18 using the following non-exclusive Media Foundation method:

### 19 MFCCreateDestinationCollection

20 The method MFCCreateDestinationCollection can be used by an application  
21 to create a group collection. This method has a pointer to the destination  
22 collection location as one input argument. In one embodiment, the destination  
23 collection includes the IMFDestination API (described above) and an  
24 IMFMediaEventGenerator API (described below in conjunction with changing  
25 destinations). This embodiment of a destination collection also includes an

1 IMFCollection API, which is described in more detail in the API section below.  
2 The IMFCollection API can be used by the application to add or remove “sub-  
3 destinations from the destination collection.

4 The application can open the content by passing the destination collection  
5 to the media engine in the “Open” call (see block 206 in Fig. 2). When the media  
6 engine makes an IMFDestination::GetOutputInfo call to the destination collection,  
7 the destination collection in turn makes GetOutputInfo calls to each sub-  
8 destination. Each sub-destination returns an output info or a collection of output  
9 infos to the destination collection. The destination collection is configured to  
10 group all these output infos into a collection and return this collection to the media  
11 engine in response to the GetOutputInfo call received by the destination collection.

### 12 **Presentation Destinations**

13 In some scenarios, the application may need to configure the destination  
14 based on the nature of the input media content. For example, application 102  
15 (Fig. 3) may need to get some information (e.g., stream identifier information)  
16 before it can configure the destination. For example, in a playback scenario in  
17 which the media content has multiple streams, application 102 may wish to assign  
18 each stream to a particular window; however, application 102 will not know which  
19 stream should be associated with which output presentation stream until it can see  
20 the presentation descriptor. The presentation descriptor gives the application the  
21 opportunity to see what streams are in the presentation before making a decision  
22 on how each stream should be outputted. The application then needs to  
23 communicate these decisions to the destination so that the destination can act on  
24 them.

1 In one embodiment, a presentation destination object (which is a  
2 subcategory of destinations) is used to address this need. In one embodiment, a  
3 presentation destination object (or more simply a presentation destination)  
4 implements an API referred to as IMFPresentationDestination in Fig. 6. In this  
5 example embodiment, the interface IMFPresentationDestination can include the  
6 following non-exclusive methods.

#### 7 GetOutputPresentation

8 The GetOutputPresentation is called by the application, and optionally  
9 includes a presentation description as an input argument. This method returns a  
10 pointer to an output presentation descriptor of the output presentation stream  
11 corresponding to the input presentation description. In scenarios in which the  
12 presentation destination is a destination collection (described below in conjunction  
13 with Fig. 7) having multiple destinations, the output presentation descriptor will  
14 contain identifying information for all of the output presentation streams  
15 corresponding to the multiple destinations. In some embodiments, if the  
16 application does not include an input presentation descriptor argument, then the  
17 method simply returns a descriptor that describes a best guess regarding the output  
18 presentation that would occur by default.

#### 19 SetOutputPresentation

20 The SetOutputPresentation is called by the application, and includes as  
21 input arguments both a presentation description and an output presentation  
22 descriptor to be matched to the stream(s) of input presentation description. In  
23 some embodiments, if the application does not specify an input presentation  
24 descriptor, then the specified output presentation descriptor is set to be the default  
25 output presentation descriptor. If the application does not specify an output

1 presentation descriptor, then the specified input presentation description should be  
2 matched to the default output presentation descriptor; such a call could be made to  
3 give the presentation destination advance notice of an upcoming presentation so  
4 that it can map out how it will match those input streams to output streams in  
5 advance.

6 The stream descriptors for the output presentation descriptor may contain  
7 metadata set by the application that indicate which input stream the application  
8 would like to be matched to this output stream.

#### 9 Example of Configuring a Presentation Destination

10 An example operational flow in configuring a presentation destination is  
11 described below in conjunction with Figs. 1 and 4. In this embodiment, the flow  
12 diagram of Fig. 4 occurs between blocks 209 and 210 (Fig. 2).

13 In one embodiment, media source provides a new presentation descriptor to  
14 the media engine to indicate that it is to begin a new presentation. In a block 402,  
15 media engine 104 can obtain a descriptor of the new presentation from the media  
16 source. Media engine 104 does so when it is to begin a new presentation. For  
17 example, in one embodiment, this occurs in response to media engine 104  
18 receiving an “Open” call from the application as shown in block 401.

19 In a block 404, in response to obtaining a new presentation, media  
20 engine 104 issues an event that it is starting a new presentation. In one  
21 embodiment, the event is defined as MENewPresentation, which contains the  
22 input presentation description.

23 In a block 406, if application 102 has signed up as a delegate (as  
24 determined in block 405) of the MENewPresentation event (typically but not  
25 exclusively when the application is using a presentation destination and wishes to

1 configure it in response to MENewPresentation), the application performs  
2 operations to get the information needed to configure presentation destination 116  
3 and match its streams to outputs. For example, application 102 can pass the  
4 presentation description to presentation destination 116 via a  
5 GetOutputPresentation call to obtain the output presentation descriptor(s) that  
6 destination 116 plans to associate with the stream(s) in that presentation. In this  
7 embodiment, the returned output presentation descriptor(s) include identifying  
8 information for the stream(s).

9 The application can then use this identifying information to configure the  
10 destination. Continuing the above example scenario of rendering multiple media  
11 streams each in a different specified window, the application can use the  
12 information obtained in block 406 to associate each of the output presentation  
13 streams to the appropriate input stream. These associations are set as properties  
14 associated with each stream on the output presentation. The application can then  
15 call SetOutputPresentation, specifying the input presentation description from the  
16 MENewPresentation event and the output presentation descriptor for each stream  
17 that contains the application-supplied matching information.

18 In a block 408, application 102 signals media engine 104 that it has  
19 completed handling of the MENewPresentation event.

20 After this point, the media engine proceeds with block 210 (Fig. 2) as  
21 shown in Fig. 4, with the presentation destination observing the matchings  
22 requested by the application. If application 102 is not a delegate for the event (as  
23 determined in block 405), then following block 404, the operational flow proceeds  
24 to block 210 as shown in Fig. 4.

1 In another embodiment, the media engine can make use of the presentation  
2 destinations without the application's involvement. It does so by calling  
3 SetOutputPresentation with the next upcoming input presentation, with a NULL  
4 output presentation. This gives the destination an advance look at the new  
5 presentation so that it can make decisions in advance about how to match each  
6 input streams for the eventual GetOutputInfo calls. This can all be done without  
7 the application's involvement.

### 8 **Changing the Destination During a Presentation**

9 In some scenarios, the destination may need to be changed during a  
10 presentation. For example, the application may decide to change the destination,  
11 or the destination may need to be changed based on the nature of the input media  
12 content (e.g., a timeline presentation in which multiple presentation streams occur  
13 in a sequence). In one embodiment, the destination includes a  
14 MEDestinationChanged event that signals the media engine that the destination is  
15 being changed during the current presentation. For example, the destination can  
16 be configured to automatically send a MEDestinationChanged event to the media  
17 whenever the destination changes.

18 In some embodiments, the destination also implements an API referred to  
19 as IMFMediaEventGenerator as shown in Fig. 6 to allow the application to initiate  
20 the MEDestinationChanged event. In one embodiment, the application can call an  
21 IMFMediaEventGenerator::QueueEvent method to queue the  
22 MEDestinationChanged event on the destination.

### 23 **Example Operation Flow in Changing a Presentation Destination**

24  
25

1       An example operational flow in changing a presentation destination is  
2 described below in conjunction with Figs. 1 and 5. In this embodiment, the flow  
3 diagram of Fig. 5 can occur anytime during the performance of the presentation.

4       In a block 502, destination 116 sends a MEDestinationChanged event to  
5 media engine 104 when the destination is changed. As previously stated, in some  
6 embodiments the destination can be configured to automatically send the  
7 MEDestinationChanged event when application 102 changes the destination. In  
8 other embodiments, after application 102 changes destination 116, the destination  
9 is configured to send the MEDestinationChanged event after the application makes  
10 a call to the aforementioned IMFMediaEventGenerator::QueueEvent method of  
11 the IMFMediaEventGenerator API. In other embodiments, the destination can  
12 send this event itself if something in its internal state has changed such that it  
13 needs to update the set of outputs.

14       In a block 504, media engine 104 then queries destination 116 for updated  
15 output presentation descriptors. In one embodiment, media engine 104 makes a  
16 call to the GetOutputInfo method of the IMFDestination API for each stream to  
17 get the updated output infos, from which media sinks can be obtained, exactly as is  
18 done at the beginning of a presentation. The media engine will then resolve the  
19 inputs and outputs, making whatever changes are needed to the topology.

### 20       **Timeline presentations**

21       In some scenarios, the presentation originating from the media source may  
22 change during the session. In one embodiment, this occurs due to a “timeline”  
23 media source that is providing a series of presentations, one after the other. Just  
24 as in the single-presentation case, the media engine will request outputs from the  
25 destination in advance of when the new presentation is to start playing. The

1 destination may hand back the same outputs as for the last presentation, or it may  
2 change one or more of the outputs (or introduce new outputs or remove old  
3 outputs) from the last presentation. Furthermore, if the destination is a  
4 presentation destination, the application can participate in this process, notifying  
5 the destination of the upcoming presentation and perhaps changing how the  
6 destination is configured in response, as described above in the section on  
7 Presentation Destinations.

### 8 **Distributed (Remote) Presentations**

9 In accordance with one embodiment, application 102 and media engine 104  
10 (Fig. 3) are also capable of presenting audio and video to devices existing on  
11 remote machines in addition to device that exist on local machines. One example  
12 where this is useful is when an application is using the above described systems  
13 while running under a remote desktop application. One particular remote desktop  
14 application, and one which serves as the basis of the example described below, is  
15 Terminal Services (TS). It is to be appreciated and understood that the concepts  
16 described just below can be employed in connection with different remote  
17 applications without departing from the spirit and scope of the claimed subject  
18 matter.

19 In the Terminal Services (TS) case, the user is typically physically sitting at  
20 the console of a TS client, while running the application from a TS server  
21 machine. Any media that is to be played, in this example, is sourced from the TS  
22 server, but needs to be sent over a network to the TS client for rendering.

23 In accordance with the described embodiment, the media engine is  
24 configured to send compressed data over the network, in distributed scenarios, and  
25 the work of decompression and rendering is conducted entirely on the machine

1 where rendering is to take place—i.e. at the TS client. This allows high-quality  
2 content to be played remotely over any particular network connection. It also  
3 ensures that the media content presented to the user goes through the same  
4 components (i.e. decoders and renderers), as it does in the regular local playback  
5 scenario.

6 In an example scenario in which remote playback can be performed,  
7 application 102 (Fig. 3) on the TS server creates a destination in much the same  
8 way it does in the local scenario. In this example, application 102 then makes the  
9 desired “Open” call on media engine 104 providing pointers to the content (which  
10 resides locally with the application on the TS server) and to the destination (which  
11 also resides on the TS server). The media engine then makes a GetOutputInfo call  
12 on the destination. The destination 452 then queries Terminal Services and finds  
13 out that the presentation is to take place in connection with a distributed scenario.  
14 Information is returned in the output infos that indicates that the media sinks are  
15 located on a remote machine (i.e., the TS client).

16 MF also creates a media engine on the TS client machine and gives it the  
17 output infos, which were obtained by media engine 104 (as described in block 208  
18 of Fig. 2) on the TS server. The TS client media engine then obtains media sinks  
19 from the output infos and uses them in performing the presentation on the TS  
20 client.

21 Regardless of whether the presentation is to be rendered locally or  
22 remotely, the destination registers with Terminal Services to receive notifications  
23 when the session changes from local to remote or vice versa. When such a change  
24 occurs, the destination receives a notification from Terminal Services, and then  
25 sends the MEDestinationChanged event to the media engine, as described in

1 another section of this document. The media engine then makes calls to the  
2 GetOutputInfo method of the IMFDestination API to get the updated information  
3 regarding where the media sinks are located, and re-configures its media pipeline  
4 to render the media to the correct location.

5 It is to be appreciated and understood that the application does not have to  
6 do anything differently between the distributed and local scenarios. The same can  
7 be said of the media source.

### 8 **Application Program Interfaces**

9 The following section provides documentation of APIs associated with an  
10 exemplary implementation of the above-described systems. It is to be appreciated  
11 and understood that APIs other than the ones specifically described below can be  
12 utilized to implement the above-described systems and functionality without  
13 departing from the spirit and scope of the claimed subject matter. This section  
14 also contains documentation of events and properties of destination objects.

15 As shown in Fig. 6, one embodiment of a destination includes the:  
16 IMFDestination, IMFPresentationDestination and IMFMediaEventGenerator  
17 APIs, which are specified in more detail below, according to one embodiment.

## **MFDestination**

The IMFDestination interface is used by the Media Engine to get to the Media Sinks to which the data from the various source streams should be sent.

In addition to the methods inherited from IUnknown, the IMFDestination interface exposes the methods LockDestination, UnlockDestination, and GetOutputInfo.

### **IMFDestination::LockDestination**

A call to LockDestination indicates that the caller is about to do a round of GetOutputInfo calls for a presentation. The caller should call LockDestination once before the round of GetOutputInfo calls and should call UnlockDestination once afterwards.

While locked, the Destination should not change anything about the outputs that it plans to hand back to the caller in GetOutputInfo.

### **Syntax**

```
HRESULT LockDestination(  
    );
```

### **Parameters**

### **Return Values**

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

### **Remarks**

If another thread calls LockDestination while the Destination is locked, the Destination should block that call until the first thread has unlocked it. If a thread has the Destination locked and calls LockDestination again, the call should allow the lock to recurse.

### **IMFDestination::UnlockDestination**

A call to UnlockDestination indicates that the caller has just finished a round of GetOutputInfo calls for a presentation. The caller should call LockDestination once before the round of GetOutputInfo calls and should call UnlockDestination once afterwards.

## Syntax

```
HRESULT UnlockDestination(  
    );
```

## Parameters

## Return Values

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

## Remarks

If another thread calls LockDestination while the Destination is locked, the Destination should block that call until the first thread has unlocked it. If a thread has the Destination locked and calls LockDestination again, the call should allow the lock to recurse.

## IMFDestination::GetOutputInfo

The GetOutputInfo method allows the caller (generally, the Media Engine) to obtain a set of IMFOutputInfos from which to create one or more Media Sinks for the given Stream Descriptor (from the Media Source). GetOutputInfo is generally called while the Media Engine is processing the IMFMediaEngine::OpenURL or OpenSource calls.

## Syntax

```
HRESULT GetOutputInfo(  
    IMFStreamDescriptor*    pStreamDescriptor,  
    IMFMediaType*           pMediaType,  
    IUnknown**              ppunkOutputInfo           );  
);
```

## Parameters

*pStreamDescriptor*

[in] Pointer to a Stream Descriptor describing the source stream. In general, this is obtained directly from the Presentation Descriptor exposed by the Media Source. Should not be NULL.

*pMediaType*

[in] Pointer to a Media Type from the list of Media Types enumerated by *pStreamDescriptor*. This indicates the Media Type that will be used by the Media Source. Should not be NULL.

### *ppunkOutputInfo*

[out] Pointer to a variable that will be set either to a pointer to an *IMFOutputInfo* or a pointer to a Collection of Output Infos. The value of *\*ppunkOutputInfo* may be set to NULL if there are no outputs for *pStreamDescriptor*. Should not be NULL.

### **Return Values**

If the method succeeds, it returns *S\_OK*. If it fails, it returns an error code. The error code *MF\_E\_NO\_MATCHING\_OUTPUTS* means that this Destination has no outputs corresponding to *pStreamDescriptor*, and in this case *\*ppunkOutputInfo* will be NULL.

### **Remarks**

Note that the Destination returns *IMFOutputInfos* and not *IMFMediaSinks*. *IMFOutputInfo* may allow direct access to a Media Sink, or it may point to an *IActivate* that will have to be activated in order to obtain the Media Sink. There is no requirement that the Destination know anything about the Media Sink that eventually gets created. This method can be used to enable remote scenarios.

Apps should NOT attempt to manipulate a Media Sink during the presentation by calling *GetOutputInfo* to get it from the Destination. This can cause undefined behavior. Instead, the app should use the appropriate service from the Media Engine. In general, the Media Engine will be the only one ever calling *GetOutputInfo*.

If the Output Info that is returned holds a Property Store, it may contain properties that will be useful in using or configuring the Media Sink that will get used.

The Media Engine can call *GetOutputInfo* at any point during the lifetime of the Destination object, and it can be called more than once for a given *pStreamDescriptor*. The Destination is not required to return the same output both times. The *IMFPresentationDestination* methods provide a stronger set of guarantees than this; please see that documentation for details.

A single Destination must not be shared between two Media Engines that are running concurrently. Some Destinations, however, can be reused after *IMFMediaEngine::Close* has been called. Destinations for whom it makes sense to reuse the outputs, such as playback Destinations, should be implemented to support reuse; Destinations whose outputs are inherently not reusable, such as archive Destinations, may fail *GetOutputInfo* calls upon reuse, or the Media Sinks returned in the outputs may simply be unusable.

The Media Engine generally calls one round of *GetOutputInfo* calls for each presentation that it gets from the Media Source (that is, a timeline source

would result in multiple rounds of calls, each for a different presentation). If the Destination wishes to specify new outputs at any point, it should implement IMFMediaEventGenerator, and it should send the MEDestinationChanged event. This will trigger the Media Engine to call a round of GetOutputInfo calls again.

It is valid for a Destination not to provide any outputs for a given *pStreamDescriptor*. In this case *\*ppunkOutputInfo* will be NULL, and MF\_E\_NO\_MATCHING\_OUTPUTS will be returned.

## IMFOutputInfo

The IMFOutputInfo interface is an interface for returning outputs from the Destination. In addition to the methods inherited from IUnknown, the IMFOutputInfo interface exposes the following methods:

GetOutput

GetPropertyStore

GetClock

### IMFOutputInfo::GetOutput

The GetOutput method retrieves a pointer to either a Media Sink or an Activateable.

#### Syntax

```
HRESULT GetOutput(  
    IUnknown **ppunkOutput  
);
```

#### Parameters

*ppunkOutput*

[out] Pointer to a variable that will receive a pointer to an IUnknown that will be either the Media Sink for this output or an IActivate object on which Activate can be called to obtain the Media Sink. Should not be NULL. Upon successful return, *\*ppunkOutput* will be non-NULL.

IActivate is a Function Discovery interface; it contains an Activate method. The function instance interface IFunctionInstance inherits from it, and sometimes the output will be a Function Instance obtained from Function Discovery.

#### Return Values

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

#### Remarks

No special remarks.

#### See Also

No special cross-references.

#### IMFOutputInfo::GetPropertyStore

The GetPropertyStore method retrieves a pointer to the Property Store associated with this output, if it exists.

#### Syntax

```
HRESULT GetPropertyStore(  
    IPropertyStore **ppPropertyStore  
);
```

#### Parameters

*ppPropertyStore*

[out] Pointer to a variable that will receive a pointer to the Property Store associated with this output, if it exists. Should not be NULL. Upon successful return, *\*ppPropertyStore* may be NULL if there is no associated Property Store

#### Return Values

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

#### Remarks

See “Destination Changes” section for properties relevant to Destinations that can be set in this Property Store.

#### See Also

No special cross-references.

#### IMFOutputInfo::GetPresentationClock

The GetPresentationClock method retrieves a pointer to a Presentation Clock that is meant to control this output independently, if one exists.

#### Syntax

```
HRESULT GetPresentationClock(  
    IMFPresentationClock **ppPresentationClock  
);
```

#### Parameters

### *ppPresentationClock*

[out] Pointer to a variable that will receive a pointer to the Presentation Clock associated with this output, if it exists. Should not be NULL. Upon successful return, *\*ppPresentationClock* may be NULL if there is no associated independent Presentation Clock; this will be the case for most scenarios.

### **Return Values**

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

### **Remarks**

This Presentation Clock should exist only if the application wanted to control the clock state of this output independently from the other outputs in the presentation. Most scenarios do not require independent clock control of outputs.

### **See Also**

No special cross-references.

## **IMFCollection interface**

The IMFCollection interface is a generic collection and enumeration interface.

In addition to the methods inherited from IUnknown, the IMFCollection interface exposes the following methods:

AddElement

GetElementCount

GetElement

RemoveElement

### **IMFCollection::AddElement**

The AddElement method allows the caller to add an element to the Collection.

### **Syntax**

```
HRESULT AddElement(  
    IUnknown* pUnknown  
);
```

### **Parameters**

*pUnknown*

[in] Pointer to an object supporting IUnknown. If *pUnknown* is NULL, then a NULL pointer gets added to the collection.

## Return Values

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

## Remarks

The pointer *pUnknown* is added to the end of the collection, so it will be enumerated last.

## See Also

No special cross-references.

## IMFCollection::GetElementCount

The GetElementCount method allows the caller to find out how many elements are in the Collection.

## Syntax

```
HRESULT GetElementCount(  
    DWORD*   pcElements  
);
```

## Parameters

*pcElements*

[out] Pointer to a DWORD in which the number of elements currently in the Collection will be returned.

## Return Values

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

## Remarks

No special remarks.

## See Also

No special cross-references.

## IMFCollection::GetElement

The GetElement method allows the caller to retrieve an element by index.

## Syntax

```
HRESULT GetElement(  
    DWORD     dwIndex,  
    IUnknown** ppUnknown  
);
```

## Parameters

*dwIndex*

[in] The index of the desired object in the Collection.

*ppUnknown*

[out] Pointer to a variable in which will be returned a pointer to the *dwIndex*-th element in the Collection. Should not be NULL.

#### **Return Values**

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

#### **Remarks**

No special remarks.

#### **See Also**

No special cross-references.

#### **IMFCollection::RemoveElement**

The RemoveElement method allows the caller to remove elements from the Collection.

#### **Syntax**

```
HRESULT RemoveElement(  
    DWORD    dwIndex,  
    IUnknown** ppUnknown  
);
```

#### **Parameters**

*dwIndex*

[in] The index of the object to be removed from the Collection.

*ppUnknown*

[out] Pointer to a variable in which will be returned a pointer to the element that has just been removed from the Collection. May be NULL.

#### **Return Values**

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

#### **Remarks**

Removing an element will cause the index of all subsequent elements in the Collection to be shifted backwards.

#### **See Also**

No special cross-references.

## Destination Events

### MEDestinationChanged

This event is queued by the Destination when it wants IMFDestination::GetOutputInfo to be called for all source streams again.

#### Remarks

Scenarios in which this event would be queued include any application- or Destination-initiated changes in the set of outputs that the Destination will return from GetOutputInfo. Examples include adding/removing outputs or changing outputs that have already been handed out. The Destination itself may not be aware that GetOutputInfo should be called again; however, if the application possesses that information, it can queue this event from the Destination directly by calling IMFMediaEventGenerator::QueueEvent.

When GetOutputInfo does get called again, the Destination will be expected to provide information about exactly what is changing. Destinations that plan on queueing this event must set certain properties. See the section entitled "Destination properties" for more details,

The event will be filled in as follows:

Event Type	MEDestinationChanged
Status	S_OK
Data Object	See remarks below

#### Remarks

If the Destination queueing this event supports IMFPresentationDestination, and if this Destination change is associated with a certain *pInputPresentation* from a former SetOutputPresentation call, the data object returned with this event should be the *pInputPresentation* pointer. Otherwise, no data should be returned with this event.

#### See Also

IMFDestination::GetOutputInfo;

Destination properties

The ppPropertyStore parameter of IMFDestination::GetOutputInfo provides a way for the Destination to specify additional data that will help in making use of the outputs that were returned.

While Destinations can define their own properties, a few common ones are defined below. All of our PROPERTYKEYs will have the FMTID MFMTID\_Destination.

MFPKEY\_DESTINATION\_StreamSinkID

Type: VT\_UI4

This property must be set for any outputs that for Media Sinks that have multiple Stream Sinks. It disambiguates which Stream Sink to use. Callers should use Stream Sink 0 if this value has not been set.

MFPKEY\_DESTINATION\_OutputID

Type: VT\_UI4

This property must be set by any Destinations that can change their set of outputs in any way. The Destination must ensure that each output has a unique ID (Destination Groups might need to reassign IDs in order to settle ID conflicts and ensure uniqueness.) It allows the caller (i.e. the Media Engine) to understand what changes are being made.

This property is required if MFPKEY\_DESTINATION\_OutputDisposition is set.

This property is optional but recommended for Destinations whose set of outputs is static.

MFPKEY\_DESTINATION\_OutputDisposition

Type: VT\_UI4

This property is optional but recommended for Destinations whose set of outputs can change. Its value will be one of the following enumerated values:

MFOUTPUT\_NEW: This Destination has never handed out this output before (i.e., this is the first time that you'll see its Output ID from this Destination)

MFOUTPUT\_UNCHANGED: This output should remain exactly the same as the last time an output with this ID was issued (i.e., if it is an Activateable, calling Activate is unnecessary if you've already done it before)

MFOUTPUT\_CHANGED: Some change needs to occur. Either the output corresponds to a different object, or it's the same object but a Media Type change needs to happen

This property provides a hint to the Media Engine about how much topology rebuilding needs to be done. If it is absent, MFOUTPUT\_NEW will be assumed.

1 MFPKEY\_DESTINATION\_MediaType

2 Type: VT\_IUNKNOWN

3 This property is optional and should be set if  
4 MFPKEY\_DESTINATION\_OutputDisposition property is set to the value  
5 MFOUTPUT\_CHANGED, and if the change involves a Media Type change to the  
6 current Media Sink. The Media Type change will be applied at the appropriate  
7 time by the Media Engine.

7 MFPKEY\_DESTINATION\_NoShutdownOnRemove

8 Type: VT\_BOOL

9 This property is optional; if missing, it is assumed to be  
10 VARIANT\_FALSE. If this property is present and set to VARIANT\_TRUE, then  
11 the Media Engine will not shut down the Media Sink obtained from this output if  
12 this output is ever not returned in a round of GetOutputInfo calls in the future.

11 *Example:* Suppose an encoding session has three source presentation  
12 segments: A, B, and C. Suppose that the application wishes to have the archive  
13 Destination present for source segments A and C, but not for segment B. The  
14 application will provide a Destination (probably configured via  
15 IMFPresentationDestination) that returns a set of outputs including some archive  
16 outputs when GetOutputInfo is called for streams from segment A and C, but the  
17 set of outputs returned when GetOutputInfo is called for streams from segment B  
18 will not include the archive outputs. However, the application does not wish for the  
19 archive Media Sink to be shut down during segment B, since it needs to stay alive  
20 for when segment C happens. This problem will be solved by setting  
21 MFPKEY\_DESTINATION\_NoShutdownOnRemove on the archive outputs when  
22 they are returned from GetOutputInfo. (The archive Media Sink will still, of  
23 course, be shutdown on IMFMediaEngine::Shutdown like everything else.)

19 MFPKEY\_DESTINATION\_RatelessSink

20 Type: VT\_BOOL

21 If this property is present and set to VARIANT\_TRUE, then the Media  
22 Sink associated with this output is treated by the Media Engine as though it set the  
23 MF\_RATELESS\_SINK characteristic from IMFMediaSink::GetCharacteristics.  
24 If this property is absent or set to VARIANT\_FALSE, then the Media Engine just  
25 uses the characteristics returned from IMFMediaSink::GetCharacteristics.

24 Media Sink ratelessness is discussed in greater detail in the Media Sink  
25 Specification and the MF Timing Model Specification. If all Media Sinks in a  
presentation are rateless, then the Media Engine does not attempt to impose any  
timing on the media pipeline, and the pipeline just runs as fast as it can.

1       *Example:* In an encoding scenario where there is a video renderer present  
2 for preview or postview, the Destination returning the video renderer as an output  
3 should set this property to VARIANT\_TRUE. That is because although video  
4 renderers are not inherently rateless Media Sinks, we want video renderers to  
5 behave ratelessly in this scenario, simply displaying frames when they arrive.

## 6       **IMFPresentationDestination Interface**

7       The IMFPresentationDestination interface is optionally supported by  
8 Destinations. For those Destinations that are able to create their Media Sinks up-  
9 front, it provides a standard way of seeing the Media Types and being able to  
10 manage the output Media Types.

11       In addition to the methods inherited from IMFDestinaton, the  
12 IMFPresentationDestination interface exposes the following methods:

13       GetOutputPresentation

14       SetOutputPresentation

### 15       IMFPresentationDestination::GetOutputPresentation

16       The GetOutputPresentation method allows the caller to view a Presentation  
17 Descriptor that represents the outputs that will be returned from GetOutputInfo.  
18 This is convenient, for instance, for apps that wish to know what Media Types will  
19 be used in the output

#### 20       **Syntax**

21       HRESULT GetOutputPresentation(  
22               IMFPresentationDescriptor\*  
23       *pInputPresentationDescriptor*,  
24               IMFPresentationDescriptor\*\*     *ppOutputPresentationDescriptor*);

#### 25       **Parameters**

*pInputPresentationDescriptor*

21       [in] A NULL value indicates that the caller wants the default output  
22 Presentaiton Descriptor in *\*ppOutputPresentationDescriptor*. Otherwise, this is a  
23 pointer to a PresentationDescriptor that has been sent to the application via the  
24 MENewPresentation event. It is not valid to use a copy of that Presentation  
25 Descriptor. The Destination will return the Presentation Descriptor that was  
associated with *pInputPresentationDescriptor* via SetOutputPresentation. If no  
such association was made, MF\_E\_DESTINATION\_UNRECOGNIZED\_  
PRESENTATION will be returned.

Must be NULL if SetOutputPresentation is not supported.

*ppOutputPresentationDescriptor*

[out] Pointer that will receive a pointer to a Presentation Descriptor describing either the default output presentation (if *pInputPresentationDescriptor* is NULL) or the output presentation associated with *pInputPresentationDescriptor*. Should not be NULL.

### Return Values

If the method succeeds, it returns S\_OK. If it fails, it returns an error code.

### Remarks

Destinations must generate their own default Presentation Descriptor (although some may allow the user to change it via SetOutputPresentation). Therefore, GetOutputPresentation( NULL, &pOutputPresentation ) can be called without ever having called SetOutputPresentation. Therefore, this call must succeed when *pInputPresentationDescriptor* is NULL.

Destinations may not know the exact mediatype of their outputs; therefore, Destinations should supply a *\*ppOutputPresentation* whose media types contain as much information as possible, even though this might be very vague information (perhaps only the major types will be valid in the media types).

### See Also

MENewPresentation

### IMFPresentationDestination::SetOutputPresentation

The SetOutputPresentation method allows the caller to indicate the desired output presentation to be handed out by the Destination. The output presentation can either be associated with a particular input presentation, or can be specified as a default presentation.

### Syntax

```
HRESULT SetOutputPresentation(  
    IMFPresentationDescriptor*  
    pInputPresentationDescriptor,  
    IMFPresentationDescriptor*  
    pOutputPresentationDescriptor);
```

### Parameters

*pInputPresentationDescriptor*

[in] A NULL value indicates that the caller wants to set the default output Presentation Descriptor. Otherwise, this is a pointer to a PresentationDescriptor that has been sent to the application via the **MENewPresentation** event. It is not valid to use a copy of that Presentation Descriptor.

In **GetOutputInfo**, the Destination will look for the *pStreamDescriptor* in all of the input presentations it has received via **SetOutputPresentation** and return an output corresponding to a stream in the associated output presentation if a match is found. If a match is not found, an output corresponding to a stream in the default output presentation should be returned. (Implementations will compare IUnknown \* pointer values to determine whether there is a match)

#### *pOutputPresentationDescriptor*

[in] Pointer to a Presentation Descriptor to be associated with *pInputPresentationDescriptor* for the purposes of **GetOutputInfo** calls. A NULL value indicates that for *pInputPresentationDescriptor* streams, the default output presentation should be used. Since a NULL value will not cause there to be any changes, even Destinations that have fixed outputs can succeed in this case.

### **Return Values**

If the method succeeds, it returns **S\_OK**. If it fails, it returns an error code. **MF\_E\_DESTINATION\_FIXED** will be returned by Destinations that do not support this functionality.

### **Remarks**

Support of this function is optional; Destinations that have a fixed output presentation can return **MF\_E\_DESTINATION\_FIXED**. (Some fixed-MediaType Destinations might want to accept this call for output presentations that match the current default output presentation if the caller simply wants to add stream-matching metadata.)

It is not valid to call **SetOutputPresentation** with a *pOutputPresentationDescriptor* that has Media Types that are unsupported by the Media Sink. Destinations should attempt to validate this whenever possible.

**SetOutputPresentation** should not be called while the Destination is locked. If the Destination is locked, implementations should either block until the Destination is unlocked or return **MF\_E\_DESTINATION\_LOCKED**.

The Destination will not automatically fire **MEDestinationChanged** when **SetOutputPresentation** is called. This is because Destinations implementing **IMFPresentationDestination** are not required to implement **IMFMediaEventGenerator**. Presentation Destinations that do wish for a change to take effect immediately should implement **IMFMediaEventGenerator**, and it

1 should be up to the application to queue the `MEDestinationChanged` event manually after making the change.

2 The *pInputPresentation* pointer should be the same one that came with the `MENewPresentation` event from the Media Engine. This is because for all calls to  
3 `GetOutputInfo` for this presentation, the *pStreamDescriptor* will be the same `IUnknown *` as those in the `MENewPresentation Presentation Descriptor`; the  
4 Destination relies on this fact in order to match streams according to this `SetOutputPresentation` call.  
5

6 In order to ensure that the application has finished making calls to `SetOutputPresentation` by the time `GetOutputInfo` gets called, the application  
7 should be a delegate for the `MENewPresentation` event, which will keep the Media Engine from calling `GetOutputInfo` until the app has signaled that it is done  
8 handling the `MENewPresentation` event.

#### 9 See Also

10 `MENewPresentation`

### 11 IMFPresentationDestination Properties

12  
13 Applications may specify the following properties in the metadata of the output presentation descriptor passed to `SetOutputPresentation`.  
14

#### 15 `MFPKEY_DESTINATION_ASSOCIATE_TO_INPUT_STREAM_INDEX`

16 The application sets this property on the metadata of the stream descriptors of the output presentation descriptor *pOutputPresentation* passed to  
17 `SetOutputPresentation` when *pInputPresentation* is non-NULL. This property should be of type `VT_UI4`. Its value should be the index (not the identifier) of the  
18 stream in *pInputPresentation* to which the stream should be matched.

19 When the Destination subsequently receives a call to `GetOutputInfo` where *pStreamDescriptor* is equal to the `IMFStreamDescriptor` pointer in  
20 *pInputPresentation*, an output corresponding to the stream on which this property was set will be returned.  
21

### 22 Example Computing Environment

23 Fig. 8 illustrates a general computer environment 800, which can be used to  
24 implement the techniques described herein. The computer environment 800 is  
25 only one example of a computing environment and is not intended to suggest any

1 limitation as to the scope of use or functionality of the computer and network  
2 architectures. Neither should the computer environment 800 be interpreted as  
3 having any dependency or requirement relating to any one or combination of  
4 components illustrated in the example computer environment 800.

5 Computer environment 800 includes a general-purpose computing device in  
6 the form of a computer 802. The components of computer 802 can include, but  
7 are not limited to, one or more processors or processing units 804, system  
8 memory 806, and system bus 808 that couples various system components  
9 including processor 804 to system memory 806.

10 System bus 808 represents one or more of any of several types of bus  
11 structures, including a memory bus or memory controller, a peripheral bus, an  
12 accelerated graphics port, and a processor or local bus using any of a variety of  
13 bus architectures. By way of example, such architectures can include an Industry  
14 Standard Architecture (ISA) bus, a Micro Channel Architecture (MCA) bus, an  
15 Enhanced ISA (EISA) bus, a Video Electronics Standards Association (VESA)  
16 local bus, a Peripheral Component Interconnects (PCI) bus also known as a  
17 Mezzanine bus, a PCI Express bus, a Universal Serial Bus (USB), a Secure Digital  
18 (SD) bus, or an IEEE 1394, *i.e.*, FireWire, bus.

19 Computer 802 may include a variety of computer readable media. Such  
20 media can be any available media that is accessible by computer 802 and includes  
21 both volatile and non-volatile media, removable and non-removable media.

22 System memory 806 includes computer readable media in the form of  
23 volatile memory, such as random access memory (RAM) 810; and/or non-volatile  
24 memory, such as read only memory (ROM) 812 or flash RAM. Basic input/output  
25 system (BIOS) 814, containing the basic routines that help to transfer information

1 between elements within computer 802, such as during start-up, is stored in  
2 ROM 812 or flash RAM. RAM 810 typically contains data and/or program  
3 modules that are immediately accessible to and/or presently operated on by  
4 processing unit 804.

5 Computer 802 may also include other removable/non-removable,  
6 volatile/non-volatile computer storage media. By way of example, FIG. 8  
7 illustrates hard disk drive 816 for reading from and writing to a non-removable,  
8 non-volatile magnetic media (not shown), magnetic disk drive 818 for reading  
9 from and writing to removable, non-volatile magnetic disk 820 (e.g., a “floppy  
10 disk”), and optical disk drive 822 for reading from and/or writing to a removable,  
11 non-volatile optical disk 824 such as a CD-ROM, DVD-ROM, or other optical  
12 media. Hard disk drive 816, magnetic disk drive 818, and optical disk drive 822  
13 are each connected to system bus 808 by one or more data media interfaces 825.  
14 Alternatively, hard disk drive 816, magnetic disk drive 818, and optical disk  
15 drive 822 can be connected to the system bus 808 by one or more interfaces (not  
16 shown).

17 The disk drives and their associated computer-readable media provide non-  
18 volatile storage of computer readable instructions, data structures, program  
19 modules, and other data for computer 802. Although the example illustrates a hard  
20 disk 816, removable magnetic disk 820, and removable optical disk 824, it is  
21 appreciated that other types of computer readable media which can store data that  
22 is accessible by a computer, such as magnetic cassettes or other magnetic storage  
23 devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or other  
24 optical storage, random access memories (RAM), read only memories (ROM),  
25

1 electrically erasable programmable read-only memory (EEPROM), and the like,  
2 can also be utilized to implement the example computing system and environment.

3 Any number of program modules can be stored on hard disk 816, magnetic  
4 disk 820, optical disk 824, ROM 812, and/or RAM 810, including by way of  
5 example, operating system 826, one or more application programs 828, other  
6 program modules 830, and program data 832. Each of such operating system 826,  
7 one or more application programs 828, other program modules 830, and program  
8 data 832 (or some combination thereof) may enact transactions, in accordance  
9 with the example embodiments described above, to implement all or part of the  
10 resident components that support the distributed file system.

11 A user can enter commands and information into computer 802 via input  
12 devices such as keyboard 834 and a pointing device 836 (*e.g.*, a “mouse”). Other  
13 input devices 838 (not shown specifically) may include a microphone, joystick,  
14 game pad, satellite dish, serial port, scanner, and/or the like. These and other input  
15 devices are connected to processing unit 804 via input/output interfaces 840 that  
16 are coupled to system bus 808, but may be connected by other interface and bus  
17 structures, such as a parallel port, game port, or a universal serial bus (USB).

18 Monitor 842 or other type of display device can also be connected to the  
19 system bus 808 via an interface, such as video adapter 844. In addition to  
20 monitor 842, other output peripheral devices can include components such as  
21 speakers (not shown) and printer 846 which can be connected to computer 802 via  
22 I/O interfaces 840.

23 Computer 802 can operate in a networked environment using logical  
24 connections to one or more remote computers, such as remote computing  
25 device 848. By way of example, remote computing device 848 can be a PC,

1 portable computer, a server, a router, a network computer, a peer device or other  
2 common network node, and the like. Remote computing device 848 is illustrated  
3 as a portable computer that can include many or all of the elements and features  
4 described herein relative to computer 802. Alternatively, computer 802 can  
5 operate in a non-networked environment as well.

6 Logical connections between computer 802 and remote computer 848 are  
7 depicted as a local area network (LAN) 850 and a general wide area network  
8 (WAN) 852. Such networking environments are commonplace in offices,  
9 enterprise-wide computer networks, intranets, and the Internet.

10 When implemented in a LAN networking environment, computer 802 is  
11 connected to local network 850 via network interface or adapter 854. When  
12 implemented in a WAN networking environment, computer 802 typically includes  
13 modem 856 or other means for establishing communications over wide  
14 network 852. Modem 856, which can be internal or external to computer 802, can  
15 be connected to system bus 808 via I/O interfaces 840 or other appropriate  
16 mechanisms. The illustrated network connections are examples and other means  
17 of establishing at least one communication link between computers 802 and 848  
18 can be employed.

19 In a networked environment, such as that illustrated with computing  
20 environment 800, program modules depicted relative to computer 802, or portions  
21 thereof, may be stored in a remote memory storage device. By way of example,  
22 remote application programs 858 reside on a memory device of remote  
23 computer 848. For purposes of illustration, applications or programs and other  
24 executable program components such as the operating system are illustrated herein  
25 as discrete blocks, although it is recognized that such programs and components

1 reside at various times in different storage components of computing device 802,  
2 and are executed by at least one data processor of the computer.

3 Various modules and techniques may be described herein in the general  
4 context of computer-executable instructions, such as program modules, executed  
5 by one or more computers or other devices. Generally, program modules include  
6 routines, programs, objects, components, data structures, *etc.* for performing  
7 particular tasks or implement particular abstract data types. These program  
8 modules and the like may be executed as native code or may be downloaded and  
9 executed, such as in a virtual machine or other just-in-time compilation execution  
10 environment. Typically, the functionality of the program modules may be  
11 combined or distributed as desired in various embodiments.

12 An implementation of these modules and techniques may be stored on or  
13 transmitted across some form of computer readable media. Computer readable  
14 media can be any available media that can be accessed by a computer. By way of  
15 example, and not limitation, computer readable media may comprise “computer  
16 storage media” and “communications media.”

17 “Computer storage media” includes volatile and non-volatile, removable  
18 and non-removable media implemented in any method or technology for storage  
19 of information such as computer readable instructions, data structures, program  
20 modules, or other data. Computer storage media includes, but is not limited to,  
21 RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM,  
22 digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic  
23 tape, magnetic disk storage or other magnetic storage devices, or any other  
24 medium which can be used to store the desired information and which can be  
25 accessed by a computer.

1       “Communication media” typically embodies computer readable  
2 instructions, data structures, program modules, or other data in a modulated data  
3 signal, such as carrier wave or other transport mechanism. Communication media  
4 also includes any information delivery media. The term “modulated data signal”  
5 means a signal that has one or more of its characteristics set or changed in such a  
6 manner as to encode information in the signal. As a non-limiting example only,  
7 communication media includes wired media such as a wired network or direct-  
8 wired connection, and wireless media such as acoustic, RF, infrared, and other  
9 wireless media. Combinations of any of the above are also included within the  
10 scope of computer readable media.

11       Reference has been made throughout this specification to “one  
12 embodiment,” “an embodiment,” or “an example embodiment” meaning that a  
13 particular described feature, structure, or characteristic is included in at least one  
14 embodiment of the present invention. Thus, usage of such phrases may refer to  
15 more than just one embodiment. Furthermore, the described features, structures,  
16 or characteristics may be combined in any suitable manner in one or more  
17 embodiments.

18       One skilled in the relevant art may recognize, however, that the invention  
19 may be practiced without one or more of the specific details, or with other  
20 methods, resources, materials, *etc.* In other instances, well known structures,  
21 resources, or operations have not been shown or described in detail merely to  
22 avoid obscuring aspects of the invention.

23       While example embodiments and applications of the present invention have  
24 been illustrated and described, it is to be understood that the invention is not  
25 limited to the precise configuration and resources described above. Various

1 modifications, changes, and variations apparent to those skilled in the art may be  
2 made in the arrangement, operation, and details of the methods and systems of the  
3 present invention disclosed herein without departing from the scope of the claimed  
4 invention.  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25